TASK-CENTERED USER INTERFACE DESIGN
A Practical Introduction

by Clayton Lewis and John Rieman

Copyright 1993, 1994: Please see the "shareware notice" at the front of the book.

Chapter 7:

The Extended Interface

The "extended interface" goes beyond the system's basic controls and feedback to include manuals, on-line help, training packages, and customer support. These are all resources that users rely on to accomplish tasks with a system.

How important is the extended interface? Isn't it true that "nobody reads manuals?" On the contrary, a survey we did recently showed that users in all kinds of jobs, from secretaries to scientists to programmers, will turn to external information sources when they don't know how to do something with a computer system. The source they turn to varies: it may be the manual, it may be the local system support people, it may be a phone support line. But users almost never know everything there is to know about the applications they use, and when they need to do something new, they look to the extended interface for help. (For details of the survey see Rieman, J. "The diary study: A workplace-oriented tool to guide laboratory studies," Proc. InterCHI'93 Conference on Human Factors in Computer Systems. New York: ACM, 1993. pp.321-326.)

Of course, the usefulness of external support varies, not only with the individual but also with the situation. For a walk-up-and use system, such as an information kiosk in a museum or a flight-insurance sales machine in an airport, the on-line interface is the whole ball game. But for a complex application aimed at a professional market, such as Mathematica or a sophisticated desk-top publishing system, external resources will be important while users are first learning the package, and they will continue to be important as a reference to seldom-used functions. In short, the use and importance of the extended interface depends on the task and the user. To accommodate this dependency, the development of the extended interface should be integrated into the task-centered design process.

This integrated design approach should begin with your very first interactions with users. You should be on the lookout for situations where manuals or other external resources would be used, as well as for situations where that kind of support would be inappropriate. Those observations will give you the background needed to rough out the design, imagining task scenarios in which users will work with both the on-line system and external support. The techniques we've described for evaluating the system in the absence of users can also be applied to the entire system, especially to manuals and on- line help. Later in the design process, when you've built a prototype of the system and the external resources, task- centered user testing with the complete package can predict much more than testing in a barren, unsupported environment that most users will never encounter.

That said, we should raise a flag of caution:

Don't rely on external support to make up for a bad on-line interface!

It's true that most users will look in a manual or ask for help if they can't figure out the system. But they don't especially want to do this, and it's not a very productive use of their time. Worse, there's a good chance that they won't find the answer they're looking for, for reasons we'll describe later in this chapter. So, you should strive for an interface that comes as close as possible to being walk-up- and-use, especially for the core functionality that's defined by your representative tasks. External support should be there as a fallback for users who stray from the common paths that you've tried to clearly define, and as a resource for users who are ready

to uncover paths that you may have intentionally hidden from novices (such as macro packages or user-programmable options).

The rest of this chapter gives guidelines for some common forms of external support: manuals, training, etc. This is a traditional division, but it's not necessarily one you should adhere to. Once again, look at your users, look at the task, and determine what would be the best way to support the on- line interface.

The Integrated Design Team

Integrating the development of the entire interface requires that the developers of all parts of the extended interface work together as a unified design team, with common goals and reporting to the same manager. This is in sharp contrast to a more traditional corporate approach that divides responsibility into software development, manual writing, and customer support. If you're in a situation where integrated development isn't an option, at least try to get individual representatives of the different areas to attend each other's meetings.

7.1 Manuals

For many users, the manual is the most readily available source of information outside the on-line interface. For other users, the first place to go for information is the on- site support person or another, more expert user -- but those experts gained much of their knowledge from manuals. In general, a good manual is the most important extension to the on-line interface.

To help understand what a "good" manual is, it's useful to imagine doing a cognitive walkthrough on a manual and noting points where the manual could fail. First, the user may be looking for information that isn't in the manual. Second, the manual may contain the information but the user may not be able to find it. Third, the user may not be able to recognize or understand the information as presented.

A task-centered design of the manual can help overcome all of these problems. If you understand the user's tasks and the context in which they're performed, you'll be able to include the information the user will look for in the manual. This will be primarily task-oriented descriptions of how to use the system itself, but it may also include descriptions of how your system interacts with other software, or comments about operations users might want to perform that aren't yet supported. Knowledge of the users will help you present this information in terms that make sense to them, rather than in system-oriented terms that make sense to the software designers. To repeat one of Nielsen and Molich's heuristics, you should "speak the user's language."

The problem of users not being able to find information that's in the manual is probably the most difficult to address. Speaking the user's language will help some, and keeping the manual "lean," including only material that's relevant, will help even more. Indeed, brevity is the touchstone of an important approach to documentation, the "minimal manual," which we discuss under the heading of training. But the problem goes deeper than that, and we describe a further solution in the section on indexing.

Deciding on the top-level organization for your manual should be another place where the principles of task-centered design come into play. Using a manual is just like using the on- line interface: people can transfer the skills they already have to the system you're designing. When you're getting to know the users and their tasks, look at manuals they're comfortable with, and watch how those manuals are used. Make sure to look at the manuals users actually rely on, which are often task-oriented books written by third parties to fill the gaps left by inadequate system documentation. Design the manual for your system so it fits the patterns of seeking and using information that users find effective.

In the spirit of borrowing, the HyperTopic in this section shows a default manual organization that should be familiar to anyone who has worked with larger applications on personal or minicomputers. The manual has three parts. First, there's a section that describes common procedures in a narrative style. This section is organized around the user's tasks. Then there's a detailed description of each command, a section that's organized more in terms of the system. Finally there's a "Super Index," a section specifically designed to overcome the "can't find it" problem that users so often have with computer manuals. We might have also included a reference card, if users expected one. As a more convenient alternative, however, we'll assume that brief reference information is included as on-line help.

The next few paragraphs describe each of the sections in the default manual organization. Keep in mind that this organization wouldn't be appropriate for every system. It would be overkill for a VCR, for example. However, many of the principles that apply to the design, such as brevity and speaking the user's language, will apply to manuals of any size.

Example: Organization of a Manual for a Large Application

The UltraProgram Manual

Detailed Task Instructions This section gives step-by-step instructions for each of the major tasks you can complete using UltraProgram. You can use these instructions as a detailed tutorial, or you can scan the headings to get an overview of a procedure.

Command Reference Here you'll find descriptions of how each menu item and dialog box in UltraProgram works. Look here if you don't understand a command's options, or if you can't figure out why it has the effect it does.

Super Index Look here if you know what you want to do, but don't know what command you should use. This is your "traveler's dictionary" that translates your phrases into the ones we use in the rest of the manual. For example, if you look up "form letters" or "address lists" or "document assembly," they'll all refer you to the sections on "mail merge" (the term used in UltraProgram).

7.1.1 The Detailed Task Instructions

The Detailed Task Instructions give the user explicit, step- by-step instructions for performing each of the major tasks the interface supports. This section will support different needs for different users at different times. Some users will work through each step of each task in order to learn the system. More adventurous users may just glance through the Detailed Task Instructions to get an overview of how a task is performed, then investigate the system on their own, referring to the instructions only when problems arise. For both novices and experienced users, the section will be used as a reference throughout the useful life of the system.

Two questions to consider in writing the Detailed Task Instructions are what information the section should include and how that information should be organized. If you've been following the task-centered design process, the question of what to include should be easy to answer: The instructions should give step-by-step instructions for performing each of the representative tasks that have been the focus of the design effort. Additional tasks may be added if the representative tasks don't cover the entire system. Training for each task should cover everything a user needs to know for that task, with the exception of things that your intended user population already knows. The cognitive walkthrough will help uncover the details of what the user needs to know, and your early user analysis should describe the knowledge users already have.

The top-level outline of the Detailed Task Instructions section will simply be a list of the tasks.

For each task, the instructions should give a brief conceptual overview of the task and the subprocedures used to accomplish it, then present sequential, step-by-step instructions for each subprocedure. The following example gives the overview for the mail-merge instructions of a word processor. That overview would be followed by the step-by-step details of each of the three major subprocedures needed to accomplish the task.

Example: Sample Overview Section of a Detailed Task Description

Section 3: Mail Merge

You can use the mail merge feature of UltraProgram to send identical or similar form letters to many addressees.

Imagine that you want to send a short letter to three customers, telling them that their account is overdue, by how many days. The detailed steps in this section show you how to: (1) create a file containing the basic letter, (2) create a file containing addresses and overdue information, (3) merge the two files to create the finished letters.

Notice that the sample overview is very, very brief. It's tempting to put a lot of detail into the overview, both to help the user understand the upcoming detailed steps and to call out useful options that the current task doesn't exercise. However, detail is exactly what the user does NOT need in the overview. If you fill the overview with technical terms and commentary on options, it will be meaningless techno-babble to the novice user. Even our simple mail merge overview won't mean much to a user who has never done something similar on another system. A novice's mental overview of an operation will slowly emerge out of an understanding of the details; the best your written overview can do is point them in the right direction.

The overview is also brief because the task itself is simple. If your representative tasks are complex, you should break them into simpler subtasks for the purpose of the manual. For example, don't give detailed instructions for doing a mail merge controlled by keyboard macros that load the files and insert today's date.

Another workable approach is to include a description of advanced options in a subsection at the end of the step-by- step task description. If you do this, be sure to put a pointer to that section into the overview, so users already familiar with similar systems can find it without working through details that don't interest them.

For each task, it's good to have a complete example, one involving actual file names, dialog box selections, etc. A more abstract description, such as, "Next, type in the file name," will inevitably leave some readers confused about the details abstracted. Showing the details of the example task will be much briefer and clearer than trying to explain the same information.

Within the step-by-step description of each task, the important principle is to be as brief as possible while still supplying the information the user needs. Remember that users already know some things, and the on-line interface provides additional information. Each step of the instructions should just fill in the gaps. Here again it's useful to think in cognitive walkthrough terms. Is the user trying to do the right thing? If it's not clear, the instructions should make it so. Is the action obviously available? If it isn't, the instructions should explain where it is. Will the user realize that the action moves them along the intended path? If not, clarify why it does. And finally, be sure to periodically describe feedback, so the user knows they're following the instructions correctly.

Writing Tips for Manuals

Be brief. Briefness is central to a usable manual. In longer manuals, users often can't find the information they need because it's buried in a mass of irrelevant text. Here are a few suggestions on how to achieve briefness: - Write a section, then go

back later and delete repetitive or unnecessary text. - Cut introductions, overviews, and summaries to the bone. Expert manual users often skip these sections, while novices just bog down in them. - Don't be historical. Users don't care about the design history. - Don't be philosophical. Users don't care about the design rationale. - Don't give technical details. Users don't care about the internal mechanisms. - Don't market. They've bought the product, don't try to sell them on it again.

Present an overview and a sequence of steps for each task. This is the approach we discuss in the text. Remember to keep the overview brief, and work through a simple but real task. A follow-up section on advanced features related to the task is optional.

Speak the user's language. You've heard this one before, but it bears repeating. You're writing a user manual, not a technical description for other software designers.

Find a light-handed editor. A good editor can catch awkward sentences and gaps in your presentation, as well as correcting errors in grammar and spelling that reduce the perceived quality of your product. A "light-handed" editor helps you maintain your personal writing style, which should reflect your knowledge of how users communicate.

Find an editor who is also a user. An editor who has a background similar to your user population can easily notice descriptions that are incomplete or hard to understand. An editor with no technical background will either limit the editing to simple grammar and spelling ("copy editing") or ask for too much additional detail in the descriptions.

Use a general and an internal style guide. A style guide gives answers to the hundreds of writing questions for which there is no "right" answer, such as whether to write "disks" or "diskettes," or whether to call your system "UltraProgram" or "the UltraProgram." A general style guide, such as the "Chicago Manual of Style," gives default answers to questions that commonly arise in books. An internal style guide is one you develop within your project or company, to cover points not in the general guide, as well as points where you want to disagree with the general guide. With an evolving internal style guide, you and your editor can make a decision on a style question once, then go on to more important things.

Use graphics sparingly and professionally. Except for screen dumps, graphics (cartoons, photos, abstract designs) usually don't add much to a manual. If you need an illustration, such as a drawing of hardware to show where switches are located, get a professional artist to do a clean, simple drawing.

Borrow an attractive and functional visual design. When you're deciding on the manual's visual design, such as the appearance of chapter headings, the font for the text, and the layout of tables, you should once again take a good look at several manuals that users find workable. You can also call in a professional designer to fine-tune things like type size and font; but borrowing from existing designs is the best way to start.

7.1.2 The Command Reference

The Command Reference section of a manual includes detailed descriptions of each command, including seldom-used information such as keyboard equivalents, hidden options, maximum and minimum sizes of data objects, etc. This information will be most useful to more experienced users who are pushing your system to its limits. It will also be turned to by system

administrators who are trying to understand problems that arise in systems they support. Many users, however, will never need the Command Reference section. It shouldn't be required to complete the reference tasks on which your task-centered design has focussed, and it is the first section you should eliminate for smaller systems.

Because the Command Reference is a system-oriented view of the interface, it may be the easiest section for you, the designer, to write. You still need to write for the user, but now you're writing for a user with more experience, so the presentation can be more technical. It's often effective to use a standard, tabular format for each command, which might list the command name, its effect, a description of options and parameters, and any warnings or limitations. Keep the entire description as brief as possible.

The organization of the Command Reference section is problematic. With command-line interfaces, the section is typically alphabetical by command name. For graphical interfaces, one option is hierarchical, following the menu hierarchy: look under File to find New, then look within that section to learn about the New dialog box. This is workable with shallow hierarchies (File...New...dialog box), but it becomes cumbersome with more deeply nested menus (File...Save...New...dialog box). For such systems, an alphabetical organization may again be more useful.

7.1.3 The Super Index

In the mid 1980s, a group of researchers at Bell Communications Research (Bellcore) noticed that it seemed almost impossible to choose the "right" names for commands. A system feature that one user called Copy, another wanted to call Move, while the designer thought it should be called Paste. This came to be known as the VOCABULARY PROBLEM. To determine how serious the problem was, the Bellcore researchers surveyed several groups of people, hundreds of individuals, asking for the names of common household objects and computer procedures. The results were very disheartening in those days of command-line interfaces.

The surveys showed that a computer system designer's first choice for a word to describe a system feature -- the "armchair design" for a command name -- had roughly one chance in ten of matching the word first assigned to the same feature by a randomly selected user. If the designer called the feature Paste, then nine out of ten users would expect it to be called something else. (G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumais. "The vocabulary problem in human-system communication," Communications of the ACM, 30 (Nov. 1987), 964-971.)

But the problem was worse than a simple indictment of armchair design. Even if the word most commonly selected by users was assigned to the system feature, there would still only be about one chance in five that a randomly chosen user would select that word. The survey of names for common objects showed that this wasn't simply a problem related to the newness of computer technology. People just didn't use the same words for things, although of course they would recognize other people's words.

Quite simply, the basic result meant that there was no "right" name for a command. Whatever word was chosen, 80 percent of the users would probably expect the command to have some other name.

Graphical user interfaces have partially overcome the vocabulary problem by asking users to RECOGNIZE commands in menus, rather than RECALL them and type them in. A user who expects a Copy menu item can usually recognize that the Move menu item has the same effect, although more complex concepts still cause problems. Within a manual, however, the vocabulary problem remains very real. A common complaint about manuals is, "I can't find anything in them."

We saw an example of this recently while watching some workers in their normal office setting. Two users were trying to find a word processor's "overstrike" feature, which they wanted to use

to mark a block of text in a document. They checked all the menus, looked in the manual, and even looked in a third-party manual. They couldn't find the entry for "overstrike" in the index or table of contents of either manual, although they were convinced they had seen the feature somewhere in the program. Finally they gave up and decided to make do with a feature that forced them to backspace and overstrike each character individually. The feature they were looking for but never found was indeed available. It could be found in a dialog box under the font menu, and it was listed in the index. It was called "strikethru."

As this example illustrates, the vocabulary problem can seriously reduce a manual's usefulness. The typical manual's index has very few entry points to each concept or command. There will be the item itself, possibly listed hierarchically under some broader heading (such as Copy under Edit). And there will be a few "see" or "see also" references that point to the same concept ("Clipboard, see also Copy"). But there won't be entries for synonyms unless those synonyms have actually been used in the text. That means that the user who is looking under "Move" may never find the section on "Copy."

The Super Index helps overcome the vocabulary problem for manuals. This index makes use of a further finding of the Bellcore researchers. A single term is clearly inadequate as an entry point into the manual, but several well chosen terms can significantly increase the effectiveness of the index.

The first step in creating the Super Index, then, is to apply one of the techniques of task-centered design: borrowing. Look at other packages in the market you're entering and determine what terms they use for various operations. You should already be using the most common of those terms as command names in your own system. Add the other terms into the index as "see" references: "Paste, see Copy" For terms that have other meanings within your system, use see also: "Move, 24, 67, 92;, see also Copy."

For larger manuals it's worth taking a second step toward creating a Super Index. Survey potential users to determine what terms they would use for system features. The survey should describe each feature without giving its name (for example, "what do you call it when text is taken away from one place and put back in another?"). Ask users to give three to six synonyms for each operation. The number of users surveyed doesn't have to be large; even a half a dozen will make a big difference, but a larger number will be better. The five to ten most common synonyms for each system feature should be added to the index as "see" or "see also" references. The results of a small experiment done by the Bellcore researchers suggested that an index based on a small user survey might make as much as a four-fold improvement in the ability of other users to find items in the manual.

7.2 On-Line Help

On-line help is the great unfulfilled promise of computer applications. Surely the power of a computer to search, reorganize, customize, and even animate information should make it possible to help the user far more effectively than with printed paper, a centuries-old technology. Well, maybe someday it will. But except for a very few products that have required extensive effort to develop, such as Bellcore's "SuperBook" and the Symbolics Lisp Machine's "Document Examiner," the attempts to deliver large volumes of information on-line have generally been shown to be no more effective than traditional books and manuals. This is the case even though the on-line systems offer hypertext features such as word search and linking to related topics. Without these features, on-line text may well be less effective. For basic on-line help systems the message is this: If you need to present more than a brief description of a system feature, don't rely on on-line help.

The ineffectiveness of lengthy on-line help files is probably the result of several factors: Text on a computer screen is usually less readable than printed text. Less text is presented on the screen than on the printed page. It's much easier to get lost while navigating through screens of text than while thumbing through pages of a book. Screens don't update as quickly as pages turn. You can't circle a word in on-line text with your pencil, or dog-ear a page. The help window or screen often covers the part of the interface that the user has questions about. And,

people haven't practiced reading on-line text for most of their life, as they have with printed text. The combined effect of these problems over-balances all the advantages that are associated with on- line text.

But while on-line help isn't the place to put the entire manual, it does have potential uses. It's an excellent place to put brief facts, "one liners," that users need frequently or are likely to forget. A prime example is the definitions of function keys or keyboard equivalents to mouse menu items. On many systems, these mappings are shown on the mouseable menus, an effective form of on-line help for users who are interested. Another example is a list of command options for a command-oriented system. Users often know exactly what they want to do in these systems, but forget the exact spelling or syntax of a command. A SIMPLE display of that information can save the user the trouble of digging out a manual. For these and other simple facts, on-line help is usually better than a reference card, which is easily lost or misplaced.

The most common failure of on-line help is to provide too much information. It's not that users could never apply the extra information, but that they often won't be able to find what's immediately relevant within the extra text. The solution is to cut the on-line text to a bare minimum. As a rule of thumb, a line of information is useful. A paragraph of text is questionable, although a table with several lines may work. An entire screen should usually be relegated to the manual.

To go a little beyond the rule of thumb, you can apply a slightly modified version of the cognitive walkthrough to on- line help. Imagine a user with a problem that the proposed on-line help could solve. Ask yourself: Will the user think of looking in on-line help? Will the user be able to find the information that would solve the current problem? And if the user finds the right information, will it be obvious that it answers the user's question? The last two points recall the vocabulary problem, which applies in spades to on-line help systems. These systems can't easily be scanned or browsed, they usually don't have an index, and reading speed and comprehension deteriorate for on-screen text.

7.3 Training

In this section we'll use the word "training" to include both classroom instruction and tutorials, on-line or otherwise, that users can do on their own. Many of the ideas about training for computer systems were developed before personal computers were widely available, when users had to be trained in the most basic computer concepts and activities. Today most users have experience with many kinds of computers, including PC's, games, telephone interfaces, automated teller machines, and so forth. Designing training for these users is both easier and harder than working from the ground up. It's easier because less training is needed, especially if you've borrowed key interface techniques from programs the user already knows. But it's harder because you have to decide what topics the training should cover, if any.

Once again, the task-centered design process should have already provided an answer to the question of what the training should cover. It should cover your representative tasks, just like the manual. In fact, the manual described in this chapter should be an effective training device for users who want to work through it. It provides exactly the information that's needed, and it's organized around the appropriate tasks. The manual also provides an appropriate fallback for users who want to explore the system without guidance.

However, users differ widely in their approaches to learning a new system. Some users, and some managers, prefer a classroom training situation, or at least a section of the documentation that is specifically designed as a tutorial, not just part of the reference manual. If you decide to support the needs of those users, the tasks described in the manual still make a good starting point. Unlike the manual, however, the training package needs to be structured in a way that will force the user to become actively involved in working with the system.

A "minimal manual" version of your basic manual can serve as an effective centerpiece for a

training program. The minimal manual, an idea suggested and tested by John Carroll at IBM, goes a step beyond the brevity we recommend for the basic manual. The minimal manual is intentionally incomplete. It couples a carefully crafted lack of details with clear task goals, forcing the user to investigate the on-line interface to discover how it works. (J.M. Carroll. "The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill." Cambridge, Mass.: MIT Press, 1990.)

Several studies by Carroll and other researchers have shown that the minimalist approach yields dramatically shorter learning times than traditional, fully guided training. The approach is also supported by several things that psychology has discovered about human learning (see HyperTopic). Some authors even recommend that the only manual supplied with a system should be a minimal manual. We think that view underestimates the value of a more complete manual for long- term reference, but we do recommend minimal training documents. And we echo the minimalist call for the briefest possible presentations of information, in training, manuals, and throughout the external interface.

Some Psychological Facts About Learning

Experimental psychologists have been studying learning and memory objectively for over a hundred years. They still can't tell you how to transfer knowledge from one person to another with the ease of copying files onto a new disk, but they have discovered some things that help predict what kinds of training will help the user learn things faster and remember them better.

In this section we describe some of the more powerful learning effects that have been discovered. You should, however, take this information with a grain of salt -- maybe even with a whole saltshaker. All of these effects have been repeatedly validated in laboratory experiments, but learning in the real world involves a complex mixture of influences, many of which aren't well understood because they are too hard to study in the laboratory.

So, use these facts as ideas to help you improve your training program, but keep in mind that they are only a small part of a much larger story.

People learn facts best if they have to work with them. There are lots of ways to make learners "work with" what they're learning. You can describe two procedures, then let them decide which is correct for a given situation. You can ask them to figure out on their own what the system does (this is called "discovery" learning). You can ask them to think about how the procedures they are learning would apply to their own work. You can teach them facts briefly, then later ask them to recall what they've learned. The important thing is to do something more than simply presenting the step-by-step procedures and expecting those to be memorized.

Skills improve with practice. Like a lot of psychological discoveries, this seems obvious. But it may not be obvious how broadly applicable it is. Essentially any skill, from typing to memorizing numbers to skiing to solving math problems, can be made faster and more accurate with the right kind of practice.

Practice requires feedback to be effective. The practice that is most effective for learning is practice in which learners are encouraged to try things and then given immediate feedback as to whether they've done the right thing or not. This kind of practice, incidentally, is a lot of work for the learner.

Spaced practice is more effective. Practicing the same simple exercise over and over (for example, saving a file) isn't as effective as doing the exercise once, then doing several other things, then doing the original exercise again. In a training program, a set of skills should be introduced, then the instruction should go on to other things. Later, the learner should be put in a situation where the first set of

skills have to be recalled and used again.

Things that shouldn't be learned should be varied. If you present a series of exercises involving files named File-A, File-B, File-C, File-D, and File-E, then the users may go away thinking that all files have to be named "File" plus a letter. (This is something we've actually seen happen, before basic computer knowledge was more widespread.) More subtly, if your system has several screens available but the training exercises all happen to be done while one screen is showing, then users will be more likely to remember what they've learned when that screen is visible in the future -- even if they realize that the same facts apply to the other screens.

People can only work on learning a few things at a time. This factor most often becomes a problem when the learning environment itself requires more attention than the material that should be learned. An on-screen tutorial that requires the user to learn several keystrokes to navigate through it is one bad example. Another is a tutorial that frequently requires the user to find and load special files from floppy disks.

People can't run (or dance) before they walk. This is as much common sense as psychology, but it interacts with the previous item about learning only a few things at a time. If learners don't have a good grasp of basic skills, such as file loading, keyboard and mouse usage, menu selection, and using the "undo" menu, then they won't be able to concentrate on learning more sophisticated skills. An effective way to teach computer skills, where possible, is to teach basic skills one day and send users back to their office for a week of practice. Then bring them back for a day of advanced training. (What's "basic" and "advanced" depends on the user and the system, of course.)

7.4 Customer-Support Phone Lines

Phone support is a huge cost for a successful product, and a good motivator for more attention to usability. Do the arithmetic by multiplying out two five-minute phone calls per customer by the number of customers: for 100,000 customers that's a million minutes on the phone, or about five years of working days. If you cut that back to one call by improving your design you're saving a lot of money directly, to say nothing of the value of increased customer satisfaction and productivity.

In setting up an effective customer support line, the three keys are training, tracking, and -- once again -- task- centered design. Training should be obvious. The people answering the phone should have the answers, or know where to get them, and they should have some basic training in phone communication techniques. Tracking refers to the user's questions, NOT the users themselves. You want to know what problems the users are having, even if they are problems that seem to be "the users' fault." Those are the problems you should be addressing in the next upgrade of your system.

And again, task-centered design. If your company already has products on the market, then you have some experience with the kinds of problems users will have. If not, you can probably imagine the categories of questions that will arise: how-to questions, software bugs, compatibility problems (with other applications, or hardware, or the operating system), questions about upgrades, etc. Imagine some specific questions in each of these categories, and then get together with someone else in your design group and act out the interaction that an imaginary user would have with your phone support technician. This is a walkthrough-like technique that can suggest modifications to both the phone-support system and the supported product, and it can also be used to train the support technicians.

Communication between a knowledgeable technician and a less- experienced user will always

be problematic, and it's even harder over the phone than in person. The HyperTopic in this section gives some suggestions for improving communication. There's nothing surprising about these guidelines; any user who has dealt with on-line help would have similar ideas. Keeping in touch with the user community after your product is on the market will suggest additional improvements.

As a final suggestion concerning phone support technicians: be one! Customer calls are a great source of feedback from users. One project at Chemical Abstracts, which sells information services for chemists, has all members of the development team, including the project manager, rotate through the office that handles customer calls. This way everybody finds out what customers are trying to do, and what problems they encounter, first hand.

Suggestions for User-Centered Phone-in Help

Speak the user's language. (Obviously!)

Be polite, cheerful, and positive. Don't be arrogant, and don't make users feel like the problems are their fault.

Give frequent feedback so the user knows that the conversation is going in the right direction. An example of phone interactions where this is typically done well is the practice of reading back order information for credit-card catalog orders.

If there's information that the user will have to provide, such as a software version or license number, make sure it's recorded where the user can find it. The start-up screen is a possibility, or under an "About this program" menu, with the information also recorded in the manual in case the system won't run.

Avoid asking for information that isn't related to the user's problem. Asking for the software license number is OK. Users will understand that only licensed software is supported. But it's wasting the user's time to ask where the package was purchased (corporate users often won't know), or what is the user's zip or area code.

Make it possible for the user to call back and get the same technician for follow-up help. After explaining the problem in detail to a disembodied voice named "Janet," the user doesn't want to call back 15 minutes later and be routed to "Carl," who not only doesn't know the problem but hasn't even heard of "Janet."

Transferring the user's call to other support people is always dangerous. Something that will help is a phone system that can support a brief three-way connection during which the technician eases the transition. Otherwise the user is put on the spot, having to explain the problem again, as well as explaining why they've been transferred.

For users, the worst-case scenario is one in which you can't transfer them at all and they have to redial, especially if they have to punch their way through a touch-tone dialog and come back into the end of a hold queue.